1. Debugging a recurive function call program.

***CODE:***

#include<stdio.h>

int fibo(int i)

{

if(i == 0 || i == 1)

{

return i;

}

return fibo(i-1) + fibo(i-2);

}

int main()

{

int i;

for(i=0;i<=10;i++)

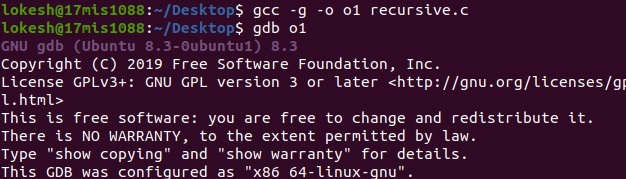
{

printf("%d\n",fibo(i));

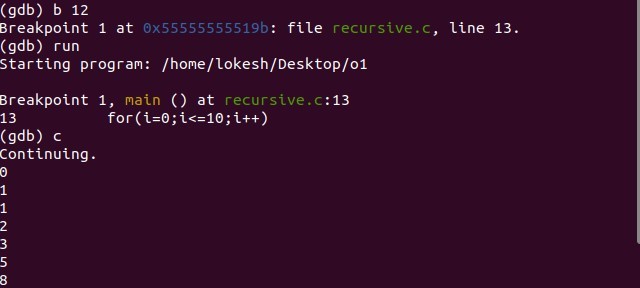
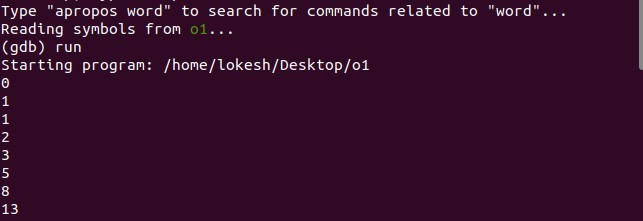
}

}

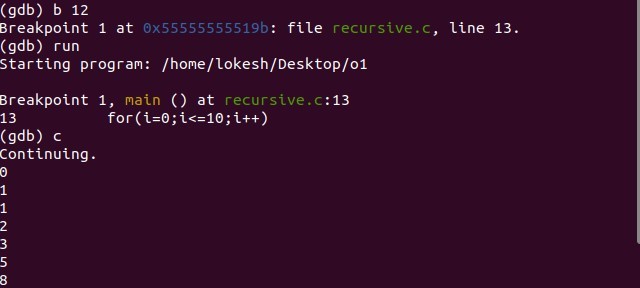
**Output:**



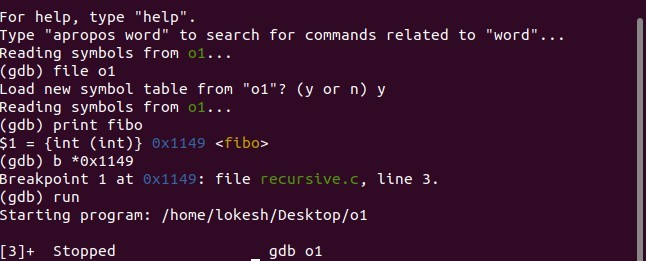
**Running gdb**



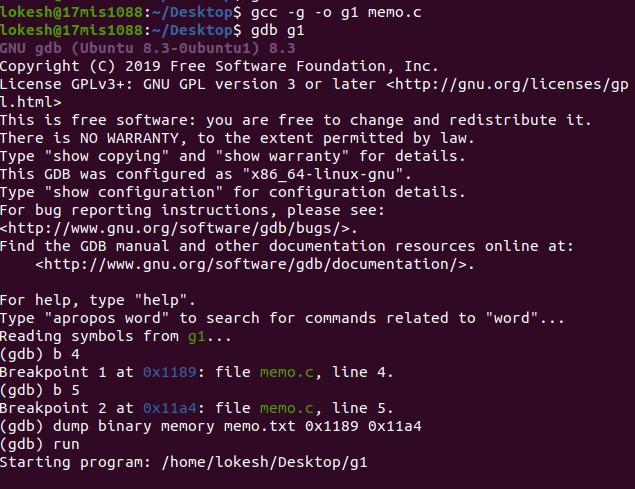
**Setting breakpoint using line number**

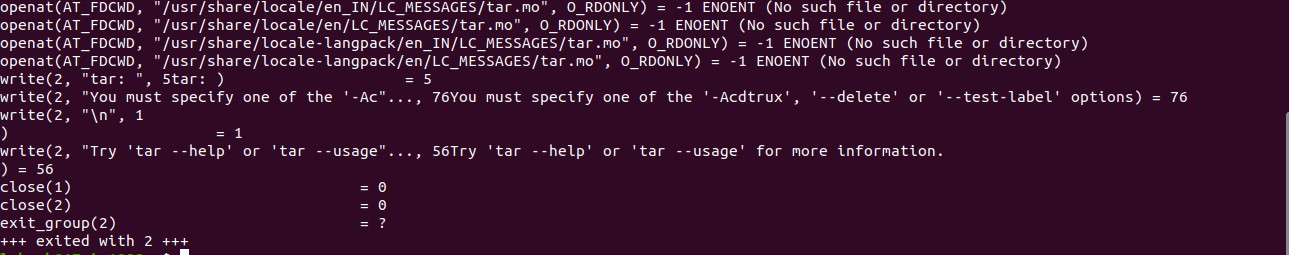
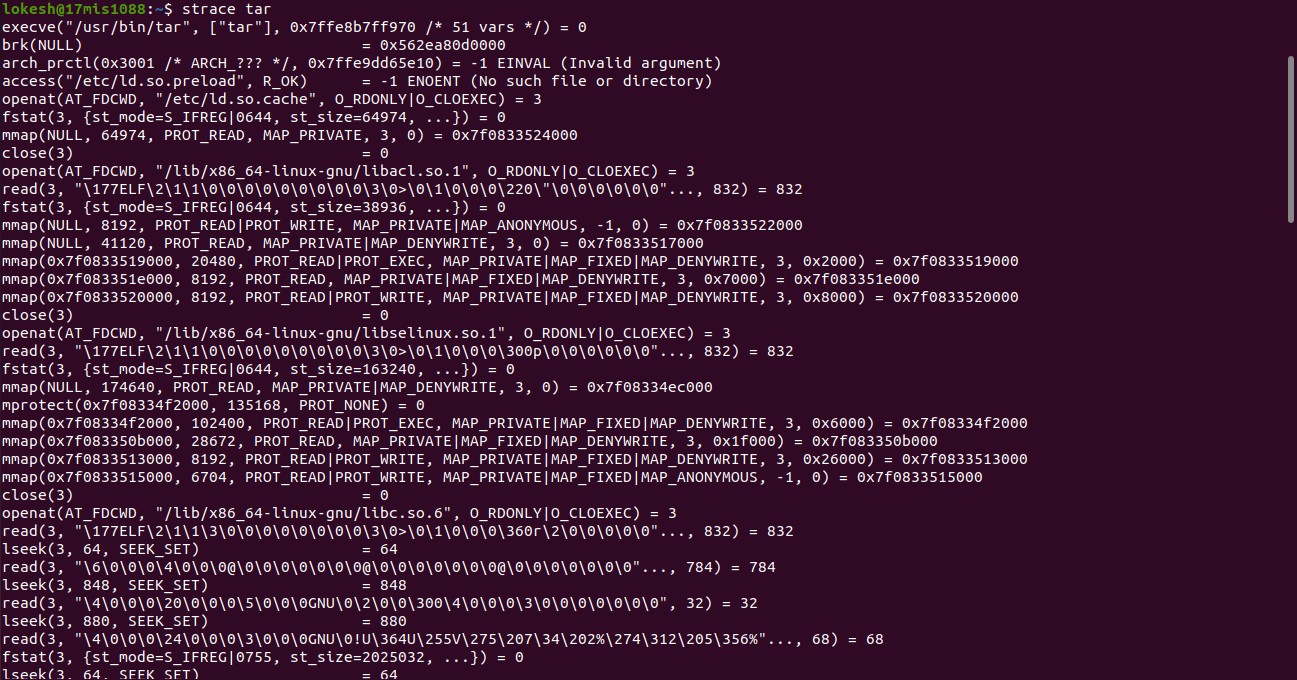


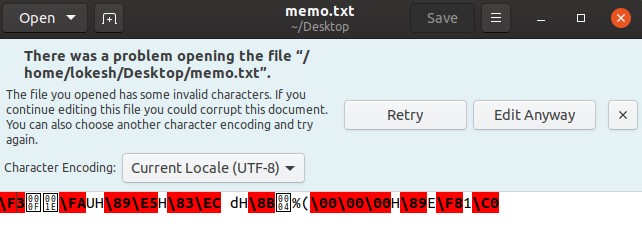
**break point using address**



**Memory Dumping**

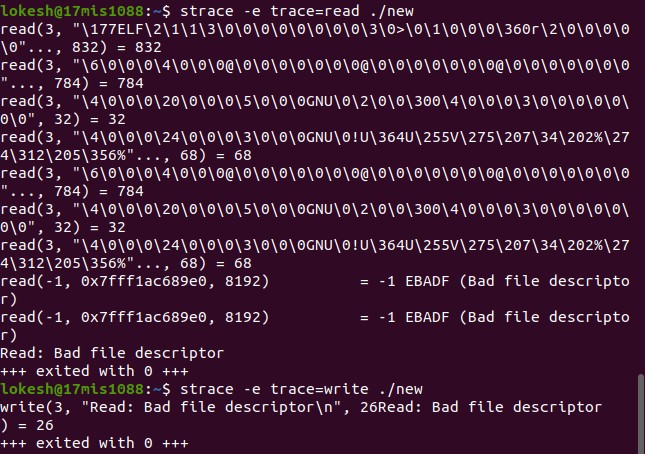


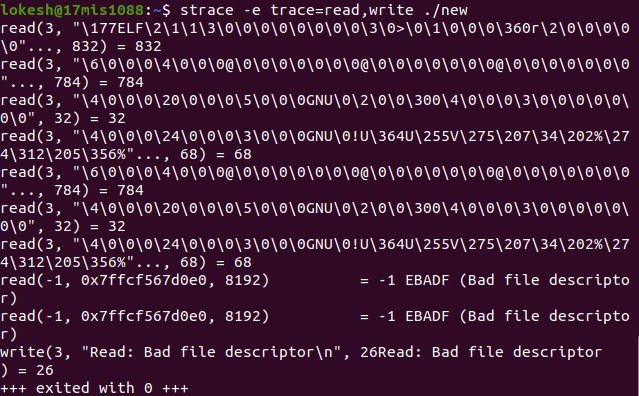




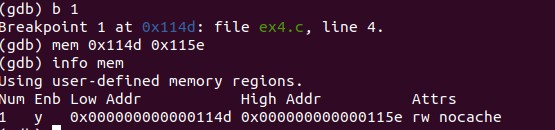
**Strace Example***To use strace first compile your program as an executable file and then type “strace filename”*

*Here: new is the executable so type “strace ./new” and also “strace -e read,write ./new” prints only system calls related to input and output.*





**Memory Region:**



**Valgrind:**

#include<stdio.h>

int main()

{

char \*p;

// Allocation #1 of 19 bytes p = (char \*) malloc(19);

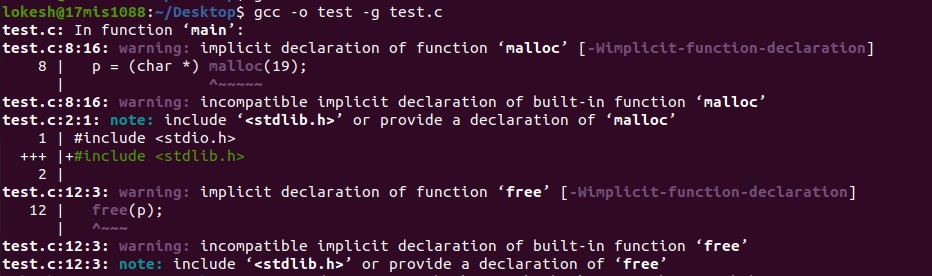
// Allocation #2 of 12 bytes p = (char \*) malloc(12); free(p);

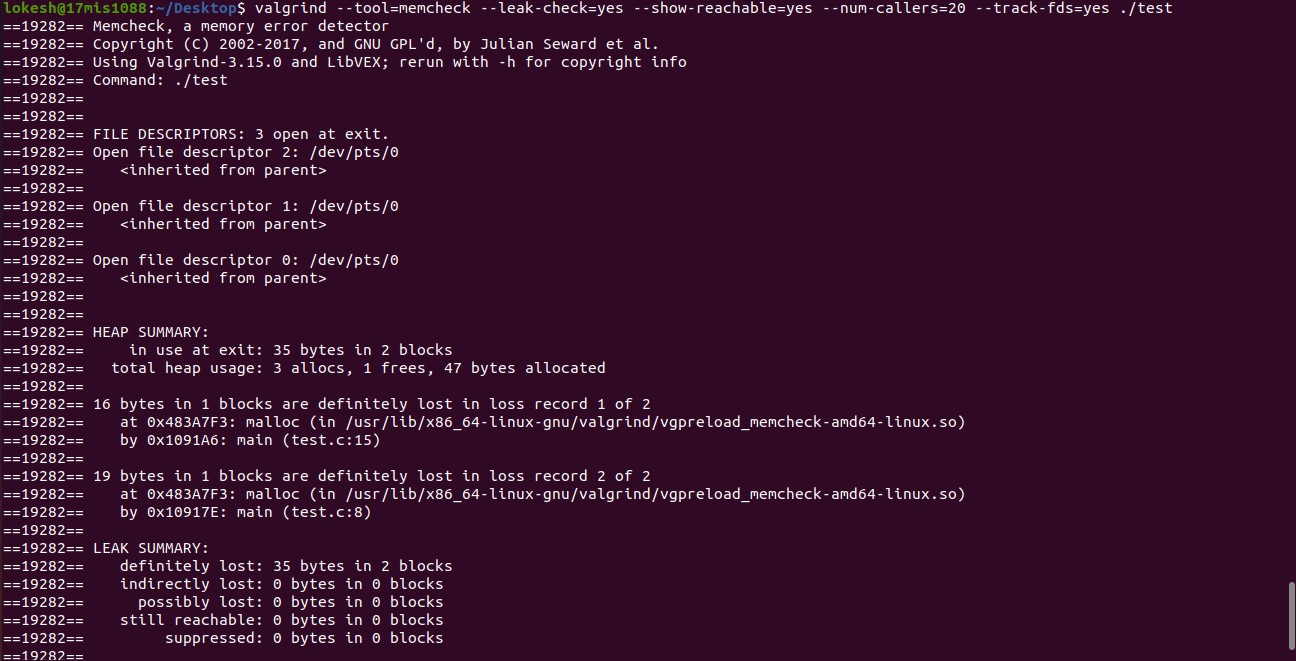
// Allocation #3 of 16 bytes p = (char \*) malloc(16);

return 0;

}

**Output**







Exercise:1 first.c :

#include <stdio.h>

void func1();

void func2(); int main() { int i=10; func1();

printf("In Main(): %d\n",i);

}

void func1() {

int n=20;

printf("In func1(): %d\n",n);

func2();

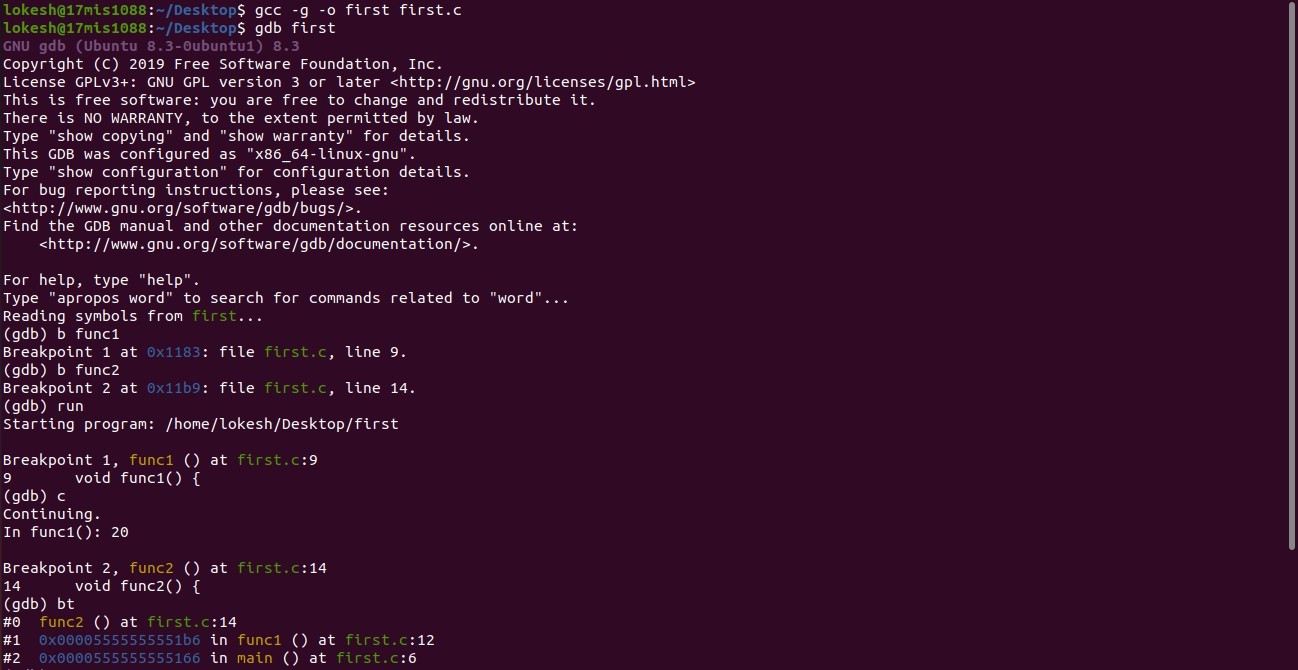
}

void func2() {

int n = 30;

printf("In func2() : %d\n",n);

}





Exercise:2

#include <stdio.h>

int sum(int n);

int main() {

int number, result;

printf("Enter a positive integer: ");

scanf("%d", &number);

result = sum(number); printf("sum = %d", result); return 0;

}

int sum(int n) {

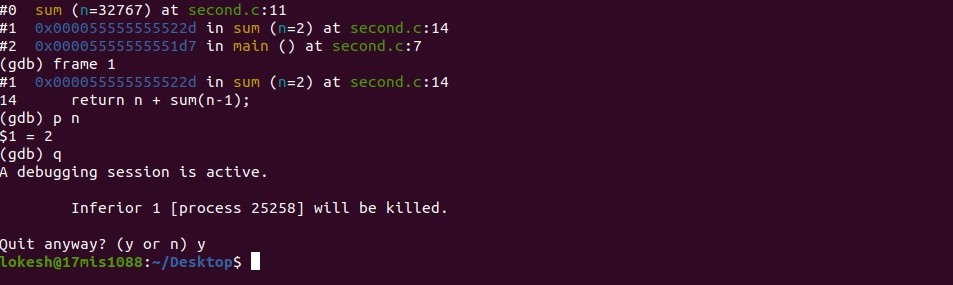
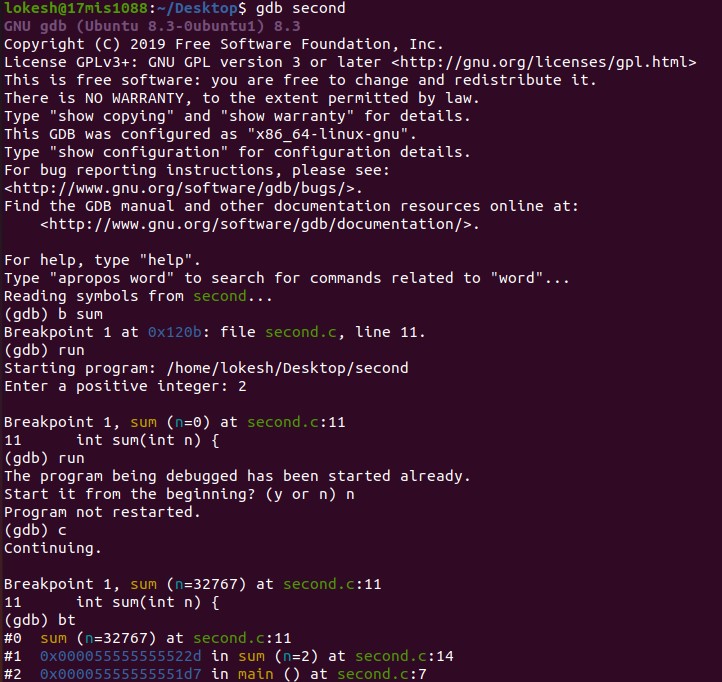
if (n != 0)

// sum() function calls itself return n + sum(n-1);

else return n;

}

**Output:**



#include<stdio.h>

int main()

{

char \*s =”Goal”; char \*t =”Home”; while(\*s++ = \*t++) printf(\*s);

return 0;

}

Exercise:3

#include <stdio.h>

/\* Print the sum of the integers from 1 to 1000 \*/

intmain(int argc, char \*\*argv)

{

int i;

int sum; sum = 0;

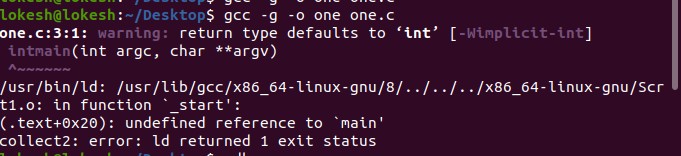
for(i = 0; i -= 1000; i++)

{ sum += i; }

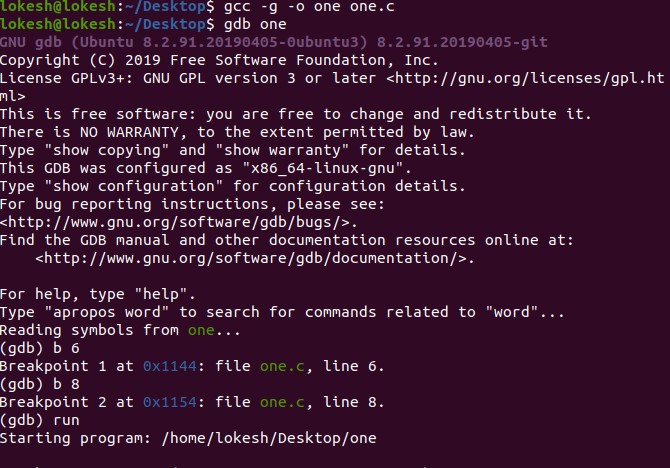
printf("%d\n", sum);

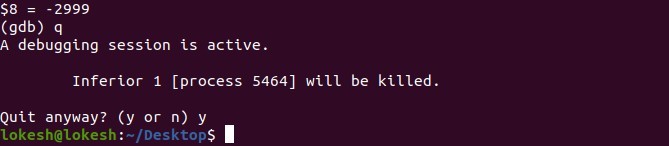
return 0;

}



There is an error in compilation so we need to correct the code the corrected code will have **int main** function correctly organised. After organizing we have





since we are getting the answer there is a logical error in the code that is

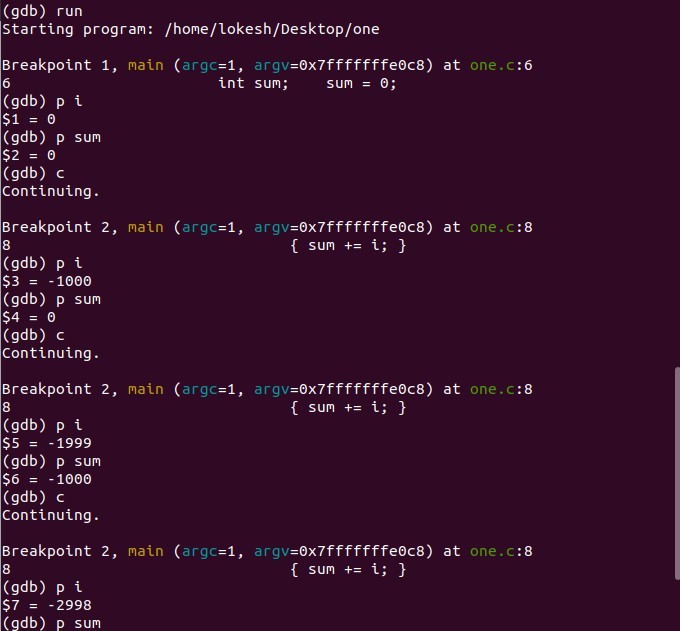
for(i = 0; i -= 1000; i++)

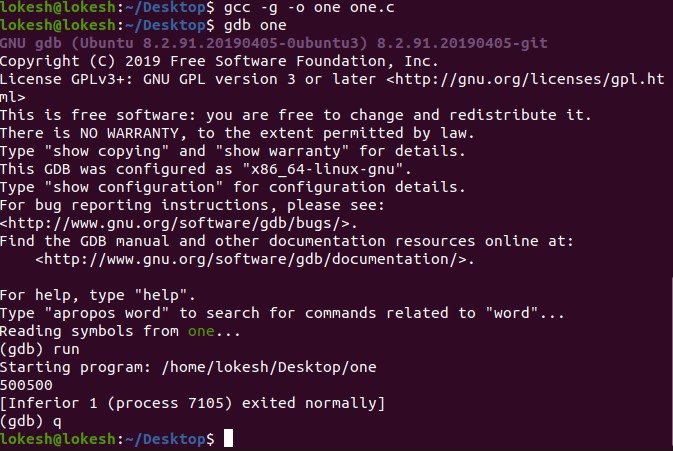
{ sum += i; }

Corrected:

for(i = 0; i <= 1000; i++)

{ sum += i; }





Exercise:4

#include<stdio.h>

int main()

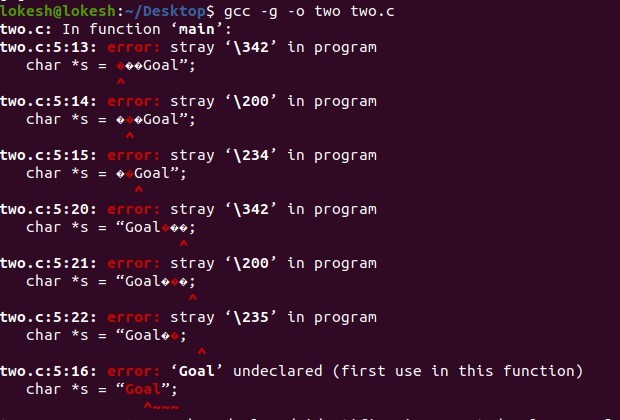
{

}

char \*s = “Goal”; char \*t = “Home”; while(\*s++ = \*t++)

printf(\*s);

return 0;



We are getting the errors because of the **“”** quatation marks are used to compile the program we need to use **""**using that

#include<stdio.h>

int main()

{

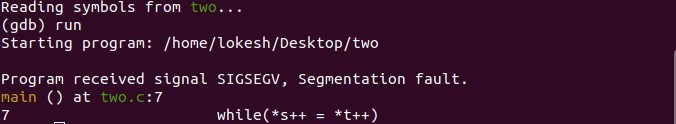
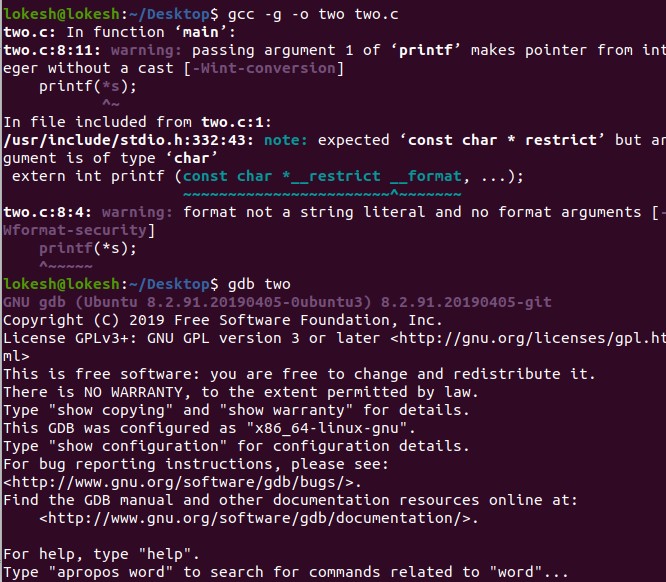
}

char \*s = "Goal"; char \*t = "Home"; while(\*s++ = \*t++)

printf(\*s);

return 0;

now we compile the code



Gdb shows the error in 7th line i.e, in the while loop. So let’s understabd the while loop first. The while loop is matching the two string literals(*s* and *t* ) and changing *s* to *t*. This modification can’t be done without causing a segmentation fault as they both are declared as string literals. So we need to change first string to char array. This way, the elements of the first string can be modified without causing a segmentation fault.

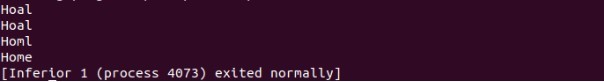
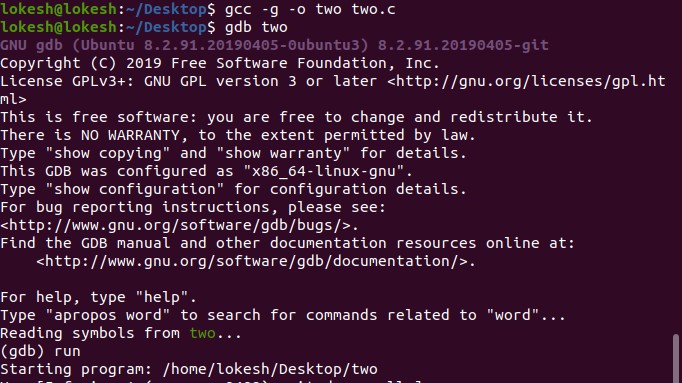
char arr[] = "Goal";

char \*s = arr;

Now, string *s* can be morphed into string *t* via the while loop without faults. The loop doesn’t need a body, and just to check if the transformation has been done properly, we print the array. So, our modified code should look like :

#include<stdio.h>

int main()



{

}

char arr[] = "Goal";

char \*s = arr;

char \*t = "Home"; while(\*s++ = \*t++); printf("%s", arr); return 0;